NPS-54-89-02

NAVAL POSTGRADUATE SCHOOL

Monterey, California

SOFTWARE MAINTENANCE:
THE NEED FOR STANDARDIZATION

Norman F. Schneidewind

February 1989

Prepared for: Navy Management Systems Support Office
Norfolk, VA 23511-6694

NAVAL POSTGRADUATE SCHOOL
Monterey, California

The research summarized herein was sponsored by the Navy Management Systems Support Office under N6856187PO30034.

Reproduction of all or part of this report is authorized.

This report was prepared by:

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
DECLASSIFICATION/DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S) NPS-54-89-02	5 MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION aval Postgraduate School — 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION
ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	7b ADDRESS (City, State, and ZIP Code)
NAME OF FUNDING/SPONSORING ORGANIZATION Navy Management Systems Support Office — 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N6856187P030034
ADDRESS (City, State, and ZIP Code) Naval Air Station Norfolk, VA 23511-6694	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO. / WORK UNIT ACCESSION NO

TITLE (Include Security Classification)
Software Maintenance: The Need for Standardization

PERSONAL AUTHOR(S) Norman F. Schneidewind

a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Technical Report	FROM Aug.88 TO Feb.89	1989 February 12	26

SUPPLEMENTARY NOTATION

COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Software Maintenance

ABSTRACT (Continue on reverse if necessary and identify by block number)

Procedures are proposed to assist the Navy Management Systems Support Office in performing software maintenance. Hardware and software maintenance are contrasted. The key difference between the two -- the ease with which software can be changed -- leads to the need for managing software change. Standardization of software is proposed as the method for managing software change. A model of software maintenance is advanced as the foundation for standardizing software maintenance.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code) (408) 646-2719 — 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete — SECURITY CLASSIFICATION OF THIS PAGE

Software Maintenance: The Need for Standardization

by

Norman F. Schneidewind

February 1989

Prepared for: Navy Management Systems Support Office
Norfolk, VA 23511-6694

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract- Procedures are proposed to assist the Navy Management Systems Support Office in performing software maintenance. Hardware and software maintenance are contrasted. The key difference between the two -- the ease with which software can be changed -- leads to the need for managing software change. Standardization of software maintenance is proposed as the method for managing software change. A model of software maintenance is advanced as the foundation for standardizing software maintenance.

I. INTRODUCTION

Software maintenance is a major activity at the Navy Management Systems Support Office (NAVMASSO). This report is provided to assist NAVMASSO in its maintenance operations. The report describes procedures for standardizing maintenance. The report makes the argument that a major attack on the maintenance problem can be made through standardization. Although the examples are oriented to local area network software and batch files, whereas NAVMASSO uses COBOL, the procedures are general and can be applied in any software development and maintenance environment.

NAVMASSO is one of the few organizations to recognize the importance of software maintenance. Most organizations emphasize development with the need for maintenance being an afterthought. NAVMASSO's concern for maintenance is exemplified by its document `NAVMASSO Data Processing Standard No. 22.A: Program Specification and Maintenance Procedures Standard', 26 August 1985. The purpose of this standard `is to describe the program design in enough detail to permit coding by the programmer and to provide the maintenance programmer personnel with the information necessary to effectively maintain the system.' Included in the standard is the objective of incorporating program maintenance procedures into the program specification. This approach integrates maintenance with development, thus forcing consideration of maintenance early in the life cycle. Equally important is the section on `Flexibility' which describes the capability for adapting the program to changing environments. Providing the capability to adapt to changing environments is the essence of the software maintenance problem and is the issue which motivated this research report: we view software maintenance as a process of change management. It is important to note that it is not only the software that changes; the documentation changes also. This important aspect of maintenance is recognized in NAVMASSO's document `NAVMASSO Data Processing Standard No. 21.B: System Documentation Development and Control Procedures', 19 June 1985 in which document changes/revisions procedures are described.

As an introduction to the subject of software maintenance, we provide some definitions followed by an explanation of the importance of the subject.

A. Definitions

Software Maintenance: Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment [1].

This definition is the conventional one and is useful if our interest i modification to software is limited to changes that are made after th software is delivered. However, it is a fact that changes are not confine to the post-delivery phase; they are made during **all** life cycle phases. I some cases, changes are made in significant numbers prior to delivery.

Maintainability: The ease with which a software can be maintained [1].

Change Management: The process of making changes to software and controlling their effects during the entire life of the software.

This definition recognizes the fact that modifications to software mus be managed effectively during the entire life of the software. It is th definition used here.

B. Cost of Maintenance

According to various sources, software maintenance accounts for significant amount of the total time and cost of running a data processi organization. For example, one study reports the following: about half applications staff time spent on maintenance, over 40 percent of the effo in supporting an operational application system spent on user enhancemen and extensions, and about half a man-year of effort allocated annually maintain the average system [2]. In another report the same authors li the factors which cause the significant maintenance effort: system ag system size, relative amount of routine debugging, and the relati development experience of the maintainers [3]. System age drives the oth factors: with increased system age, system size increases, leading greater effort allocated to routine debugging, and with increased syst age, the relative development experience of the maintainers declines due organizational turnover and change. All of these factors tend to increa the time and cost of performing maintenance. Thus maintenance is an ar that deserves a lot of attention. Improvements in maintenance practic should result in reduced costs and increased effectiveness of performi maintenance.

However there is a limit to reducing cost and increasing effectiveness through improved practices, because the maintainability of the software has largely been determined by the developer before it ever reaches the maintainer. The maintainer can only influence quality during the maintenance phase of the software life cycle. The quality of the software **as designed** is determined, in part, by whether the software development methodology assists the developer in producing maintainable software. Consequently, maintenance practices, which maintainers control, and development methodology, which developers control, are candidates for standardization.

C. Limits of Approach

The objective of standardization **is** to improve the maintainability of both existing and future software. Contrariwise, there are certain aspects of the `maintenance problem' that the above approach does not address. These are the following: 1) Much of the software that is maintained was developed without benefit of **any** methodology; consequently, methodology is not an issue in these cases; 2) Methodology is only an issue for **future** software; thus improvements in maintenance practices are only applicable to existing software; 3) An important determinant of the maintainability of software is the knowledge and skill of the developer and maintainer; 4) There are other aspects of a development methodology, such as expressiveness, that are important when evaluating it for use in addition to its usefulness **as an aid** for producing maintainable software. These aspects are beyond the scope of the paper as are the areas of software engineering environments and tools, which can contribute significantly to the quality of both development and maintenance.

The paper consists of the following sections:

o Purposes
o Objectives of Maintenance
o Metrics for Maintenance
o Model of Maintenance
o Standardization of Change Documentation
o Software Communication Mechanisms and Maintenance
o Standardization through Examination of Development Methodologies
o Example
o Further Research
o Summary

II. PURPOSES

The purposes of the paper are the following: 1) Provide a brie introduction to software maintenance by describing its objectives processes and tasks, contrasting it with hardware maintenance for th benefit of readers who may be more familiar with hardware maintenance ar 2) Present the case for standardizing software maintenance practices ar those aspects of software development methodology that affect th maintainability of the delivered software. Purpose 2 is derived from 1 c the basis that the kind of **discipline** and **rigor** that exists in hardwar maintenance should be an **objective** of software maintenance.

Notice that we do not contend that identical **methodologies** or **procedure** should be used for software maintenance because there are differences i characteristics and complexities between the two; these differences **ar** described in the paper. Rather, we propose that software maintenance shoul be supported by a model of maintenance and a minimum set of standardize practices, which would be augmented or tailored according to the needs c individual organizations or applications. The maintenance model include characteristics of development methodologies because, as stated previousl these characteristics affect maintainability.

III. OBJECTIVES OF MAINTENANCE

The objective of maintenance is to make required changes in software i such a way that its value to users is increased. Required changes c result from either the need to correct errors or to increase th functionality of the software.

A. Maintenance Process

In the broad view of maintenance, it is not limited to maki post-delivery changes [4]. Rather, it is a process that starts with us requirements and never ends [5]. Even the installation of and changes to replacement system can be considered part of the maintenance proces Our approach to identifying the maintenance functions which should standardized is to: 1) Adopt the view that maintenance is a process change management and 2) Identify tasks in maintenance that are concern with making changes to software, including changes to documentati (e.g., specification, design, listing, test plan, etc.).

B. Maintenance Tasks

Using the concept of change management, the following maintenance tasks can be identified:

- o Identify need for change
- o Determine whether change should be made, based on benefit-cost analysis
- o Evaluate the effects of change, including possible side effects
- o Determine whether change can be made without creating an incompatibility with the rest of the software
- o Make the change, if warranted, and only if it can be done in a standard way

C. Differences Between Hardware and Software Maintenance

Whereas failures in hardware are true failure events, which are caused by physical phenomena -- wearout, burnout, malfunction, or stress -- software 'failures' are error discovery events, which are caused by errors made by humans. Software errors are caused by the following: inadequate or misunderstood specifications, incorrect program logic, misuse of programming language, and mistakes in clerical operations. These errors exist in software prior to its execution and are only discovered by virtue of an input forcing the software through an execution path that contains an error.

The ability to understand the nature of errors when maintaining software has been reported to be related to the quality of documentation [6]. Therefore the characteristics of documentation that affect maintenance should be a part of any plan to improve maintenance. Documentation for maintenance is discussed in the section 'Standardization of Change Documentation'.

1) Spare Parts

For software, there are no spare parts for replacing a module that has an error. The error must be fixed before the operation can continue. This is an inherent factor which makes software less reliable than hardware.

Repair times and down times can be very long. This situation demands eas
maintainability. In particular, traceability must be achieved: the abilit
to easily trace through all relevant documents, organizations and personne
for the purpose of locating information which will assist the maintainer i
correcting the error in such a way that the change will not damage anothe
part of the program that is working (ripple effect).

2) Prototyping

Prototyping of software is similar to the hardware engineer's test benc
and development systems (e.g., in circuit emulation systems). With th
software prototype we want to obtain a quick and inexpensive test of
development idea before committing a lot of time, personnel and money t
the production system. Another objective is to test design approaches in
simplified and controlled environment without the confounding interactio
of a large system present. If the ideas won't work in the prototype, ther
is no hope of them working in the production system. One use of a prototyp
seldom mentioned is to test for flexibility of making changes to th
software. For example, is the software constructed so that the effects c
making changes are highly visible?

In many cases the prototype is treated as throwaway code. It is used fo
the purposes described and an improved version, based on the lesso
learned, is coded as the next prototype or as the production system, wh
the design iteration process ends.

IV. METRICS FOR MAINTENANCE

In order to manage software change it is desirable to measure t
effects of change. This is accomplished with quality metrics. A quali
metric is defined as follows: a quantitative measure of the degree to whi
software possesses a given attribute that affects its quality [1]. Ideall
there would be agreement on a set of application-independen
language-independent, software structure-independent metrics ('univers
metrics'). Agreement does not exist in the software engineering communi
on a universal set. Lacking this agreement, metrics which are known to
related to the effectiveness and efficiency of the software developme
process are used during development to measure and improve the developme
process; these are called process metrics [7]. It is assumed that their u
will result in maintainable software. However, process metrics, li
traceability, have little to do with measuring whether the system achiev
its quality requirements. For that we need product metrics li
reliability, accuracy, response time, throughput, etc. The two types
metrics are related in the sense that high process metric values wi
contribute to high product metric values. Product metrics are beyond t
scope of this paper.

The role of metrics in maintenance can be demonstrated by posing the following question:

When a maintenance action is taken, how are the relevant metrics values affected?

- o What are the relevant metrics?
- o What were the original values?
- o What are the new values?
- o Examine incremental changes
  - * Are they in the right direction (e.g., reduced complexity)?
  - * Are they approximately the right values (e.g., within the bounds of experience with respect to the maintenance action)?

V. MODEL OF MAINTENANCE

To explain the dynamic interaction between development and maintenance as exemplified by the changes in metrics values as a result of development and maintenance actions, the model in Figure 1 is provided. A model of the maintenance process is essential for standardization to be achieved. Different organizations may want to use different metrics, depending on the relevance of the metrics to their maintenance environments and projects.

```
                         |-------------------|
                         | Development       |
                         | Methodology       |
                         |-------------------|
                              |         |
               Contributes    |         | Affects Ability
               to Original    |         | to Make Correct
               Metrics Values |         | Changes
                              |         |
               |--------------|         |--------------|
               |                                       |
               |                                       |
               v           New Metrics                 v
     |-------------------| Values            |-------------------|
     |Compute & Recompute|------------------>| Maintenance       |
     |Common Metrics     |<------------------| Actions           |
     |-------------------| Changes Metrics   |-------------------|
               |           Values                      |
               |                                       |
               |                                       |
               v                                       v
          (Sample List)                               Add
          Completeness  -|                            Delete
          Consistency    |                            Modify
          Modularity     |---> Improved                |
          Traceability   |     Maintainability?        |
          Verifiability -|                             |
               |                                       |
               |                                       |
               |                                       |
               v                                       v
     |-------------------|   |-----------|   |-------------------|
     | Metrics Data Base |   |           |   | Maintenance Data  |
     | (Metrics and      |-->|Correlation|<--| Base (Maintenance |
     |  Projects History)|   |     ?     |   | Action History)   |
     |-------------------|   |-----------|   |-------------------|
```

Figure 1. Model of the Interaction between Development, Maintenance and Metrics.

This model may be understood and applied as follows:

A. Evaluate: Estimate the incremental change in metric value of a proposed maintenance action. If the software change is made, measure its effect after the change is made. To the extent feasible, quantify the effect of the change. The following questions are relevant when considering a change to software:

o Given the development methodology and a maintenance action, how will the metrics values be affected (magnitude and sign)? Will they change in a direction to indicate the software will be (or has been) improved? Or will the change indicate that the software will be (or has been) degraded?

This model would assist the maintenance organization to: 1] determine whether a change should be made, 2) determine whether a change improved maintainability, if it was made, and 3) document the history of the project and the change so that this information can be used when making future change decisions.

B. Feedback: Understand that taking a maintenance action changes metrics values and that the new metrics values will influence future maintenance actions.

C. Data bases: Maintain data bases of project characteristics, metrics, and maintenance actions as an aid to learning from the past: Was a given metric a good predictor of the effect of a given maintenance action? Which maintenance actions improved and which degraded the software for given project characteristics? Did the nature of the development methodology influence the maintainability of the software?

VI. STANDARDIZATION OF CHANGE DOCUMENTATION

Because there is a great difference in applications, programming environments, etc., in various organizations, the maintenance standard should accommodate those differences and specify only a minimum set of requirements and procedures.

Standardization can be viewed as a process of posing questions prior to a maintenance action and having the maintainer answer them. The purpose of this is to ensure that the maintainer has thought about the consequences of proposed changes and is alerted to potential pitfalls. Maintenance decisions and actions should be recorded in a data base for use in making future maintenance decisions.

The entities which are subject to change are software components (a element of a software system such as a module or unit). For the sake o brevity, `software component' will hereafter be called `component'.

A. Documenting the Effects of Change

It should be a standard procedure of maintenance to document a propose change in the following format (or similar format) and, if the change i made, to fill in as much detail as possible about the change. The items t be considered in deciding on a change are more important than the specifi format used to document the change. The Xs in the matrix indicate relationship between an input item and an output item.

Change an input

Type

Format

Value (How are outliers handled?)

Range

Precision

Accuracy

Name (Standardize name; should say what module does)

Questions:

* What is the effect of input on outputs?

* What is the effect of input on computation of function?

  * Computation within bounds?

TABLE I

EXAMPLE INPUT-OUTPUT CHANGE RELATIONSHIP

	OUTPUT (Name)					
INPUT (Name)	Type	Format	Value	Range	Precision	Accuracy
Type	X					
Format		X				
Value			X	X	X	X
Range			X	X	X	X
Precision			X	X	X	X
Accuracy			X	X	X	X

B. Documentation Requirements

As a minimum the following should be standard documentation fo supporting maintenance: requirements specification, design specification program listing, test plan, and test results, as summarized below.

Phase	Documentation
Requirements Analysis	Requirements Specification
Design	Design Specification
Coding	Listing
All	Test Plan, Test Results

VII. SOFTWARE COMMUNICATION MECHANISMS AND MAINTENANCE

Mechanisms which are available for communicating between components ar an important aspect of maintenance because of the serious consequences o making an error in adding or changing a linkage. As opposed to other type of software changes, a change in a communication mechanism affects mor than one component. This is particularly important for networks where defective mechanism can adversely affect the operation of computers a remote sites.

A. Kinds of Communication Mechanisms

- o Data linkages (for the transfer of data)
- o Control linkages (for the transfer of control)
- o Subroutine call
- o Procedure call
- o Message passing
- o Remote procedure call (RPC)
- o Transaction (e.g., update in a data base management system)

B. Characteristics of Communication Between Software Components

1) Explicit: There is an actual transfer or exchange of data or passing of parameters or an output from one component is the input to another component.

2) Implicit: Based on the position of the given component within a sequence of components (e.g., instructions in a program)

Before components are added, deleted or modified, it should be standard procedure to ascertain and document the effects of making the change on inter component communication. Furthermore, if the change is made, as much detail as possible should be documented about the change, as suggested by the questions below.

3) ADD a component

   o What other components will the given component communicate with once it is added?

   o What are the communication linkages? (parameter passing, message exchange, RPC, etc.?)

   o What existing communication linkages will be affected by the change?

4) DELETE a component

   o What communication linkage will be broken by the deletion?

   o What are the new communication linkages that result from the deletion?

5) MODIFY a component

   o What is the existing communication linkage which involves this component?

   o How will this communication linkage be modified by the change in the component?

VIII. STANDARDIZATION THROUGH EXAMINATION OF DEVELOPMENT METHODOLOGIES

There is evidence that the characteristics of development methodologies [8] and the characteristics of programming languages [9] can influence maintainability.

A. Characteristics of Development Methodology

When we maintain software we may not be cognizant of the developmen methodology which was used to produce the software, but it will affect ou ability to maintain the software. The evaluation hinges on a singl criterion: **does the methodology support the creation of software which i easy to change without inducing side-effects (an unexpected and undesirabl result of making a change ?).** This objective will be achieved if th methodology forces the designer to formally consider the consequences o making a change once the software has to be maintained. It follows that i order to capitalize on a methodology that supports maintenance, it i necessary to use that methodology to maintain the software. The followin is a standard procedure for evaluating a methodology with respect to it capability to support maintenance.

Does the methodology assist to:

1) Prevent side effects when performing maintenance

2) Provide ability to make selective change (i.e., don't change or destroy another part of the software when making a change)

3) Reduce dependencies between inputs, processes and outputs (dependices make it difficult to change the software without affecting something else which was working correctly prior to the change)

4) Determine whether change can be made without creating an incompatibility with the rest of the software

5) Support a rational change policy:

   o Make a change, if warranted, and only if it can be done in a standard way, a `standard way' being defined as being in conformance with the above procedure for assessing the impact of change.

   o Keep changes small

   o Make changes in small, controlled increments

   o If there is a big change to make, break the changes into manageable pieces.

IX. EXAMPLE

A. Characteristics of Development Methodology

The process of identifying and evaluating development methodology principles that are conducive to maintenance is illustrated with real examples from personal computer network operating system software (IBM PC DOS V3.2 [10] and PC LAN Program V1.1 [11]) and the state diagram method of specifying software logic [12].

A batch (command file) for starting a user personal computer on a local area network (LAN) and assigning resources provided by a server is shown in Figure 2 and the corresponding state diagram is shown in Figure 3. This batch file was modified to provide some additional network capabilities as shown in Figure 2; the corresponding modification is shown in Figure 3 with dotted boxes. The boxes represent states and the arrows represent state transitions. The numbers on the left side of the commands in the batch file correspond to the numbers on the state boxes on Figure 3. The convention for labeling state transition arrows is: Event/Action. In some cases in Figure 3 there is no event; in these cases `NE' is used to indicate this. The DOS and PC LAN Program handle transfers of control implicitly (e.g., a transfer of control occurs automatically from PC LAN Program to DOS under certain error conditions). There is no capability in the batch file language for describing error conditions explicitly, although they are shown in the state diagram to clarify the operation.

Asterisks in the batch file identify comments. Unfortunately, the comment concerning accessing the D drive was not changed with the modification. This comment is no longer applicable and caused confusion in trying to understand the program logic. With the modification, neither the D drive nor the directory program 1DIR are accessed at this point in the program. The comment should have been changed to refer to the E drive and the PROFILE program. This affects the transitions from states 5 to 6 and 6 to 7. For the sake of brevity, the error events and actions associated with states 6′ and 7′ are not shown in Figure 3; they are similar to those for states 6 and 7.

Neither a state diagram nor another type of methodology that would sho the consequences of making a change was used in creating the batch program The use of such a methodology would have helped to avoid this kind of erro by:

o Preventing side effects (erroneous comment)

o Providing ability to make selective change (replace commands 6 and 7 wit 6′ and 7′ correctly).

o Identifying existing communication linkages (communication betwee commands 6 and 7 and the D drive and its directories) and by identifyin changed communication linkages (communication between commands 6′ and 7 and the E drive and its directories).

```
: *** Start.Bat = Start.TU3
: *** For Token  Ring  User.  User  3270  Emulation on User Hard Disk
: *** Loads Profile Which is Used for Checking Hardware Compatibility
1 ECHO OFF
: *** Establish Path to Network and DOS Programs Residing on User
:     Computer
2 PATH C:\NETWORK;C:\APPS\DOS
: *** Establish Access for 1DIR to 1DIRDATA Sub Directory
  APPEND C:\1DIRDATA
  ECHO ON
: *** Load Token-Ring Programs
3 TOKREUI
  NETBEUI
: *** Start the  User Computer on the Network, Using Name Provided by
      User
4 NET START MSG %1 /SRV:1 /ASG:10 /PB1:16K /USN:3 /CMD:12 /SES:18
: *** Request Use of Server Directories and Printer
5 NET USE E: \\TN3\APPS
  NET USE D: \\TN3\DISKD
  NET USE LPT1 \\TN3\PRINT
: *** Access D  Directory which Contains 1DIR and Program Batch Files
6 D:
  *** Load 1DIR
7 1DIR
```

....

```
Modification: Replace commands 6 and 7 above with commands 6' and 7':
              (comment was not changed)

:  *** Access D Directory which Contains 1DIR and Program Batch Files
6' E:
: *** Load Profile
7' PROFILE
```

....

Figure 2. Batch file for Token-Ring LAN User Computer Start Program

```
     |---------|  _______________
 0   v         |  LOAD START FILE
|---------|----|
| IDLE    |-----------------------------| CAN'T LOAD START
|---------|                             | FILE/ERROR MSG.
     |                                  |
     |  NE/LOAD START FILE FROM DOS     |
 1   v                                  v
|---------| CAN'T LOCATE           |---------|
| START   | DIRECTORIES/ERROR MSG. | BACK AT |
| FILE    |----------------------->| DOS     |
| LOADED  |                        |         |
|---------|                        |---------|
     |                               ^    ^
     |  NE/EXECUTE PATH &            |    |
     |     APPEND COMMANDS           |    |
 2   v                               |    |
|---------| CAN'T LOAD TOKEN-RING    |    |
|DIRECTOR-| PROGRAMS/ERROR MSG.      |    |
|IES      |--------------------------|    |
|LOCATED  |                               |
|---------|                               |
     |  NE/LOAD TOKEN-RING                |
     |     PROGRAMS                       |
 3   v                                    |
|---------| CAN'T START NETWORK/ERROR MSG.|
| TOKEN-  |                               |
| RING    |-------------------------------|
| PROGRAMS|
| LOADED  |
|---------|
     |  NE/START NETWORK
     |
 4   v
|---------| RESOURCES NOT
|         | AVAILABLE/ERROR MSG.
| NETWORK |-------------------                   |---------|
| STARTED |                  |  CAN'T LOAD 1DIR/ |         |
|         |                  |  ERROR MSG.       | AT D    |
|---------|                  |  ---------------->| PROMPT  |
     |  NE/REQUEST RESOURCES |  |                |---------|
     |                       |  |
 5   v                  6    v  |                 7
|---------| NE/ACCESS  |---------| NE/LOAD       |---------|
|         |   DRIVE D  |         |   1DIR        |DIRECTORY|
|RESOURCES|----------->| DRIVE D |-------------->|PROGRAM  |
|ASSIGNED |            | ACCESSED|               |(1DIR)   |
|         |            |         |               |LOADED   |
|---------|            |---------|               |---------|
     |  |                            |-------|
     |  |  DRIVE NOT DEFINED/ERROR MSG. | AT NET|
     |  |------------------------------>| MENU  |
     |                 6'              |-------|         7'
     |                 ...........               ..........
     |     NE/ACCESS   .         . NE/LOAD       . PROFILE .
     |       DRIVE E   . DRIVE E .   PROFILE     . PROGRAM .
     ------------------>. ACCESSED.------------->. LOADED  .
                       ...........               ..........
```

Figure 3. State Diagram of a Token-Ring LAN User Computer Start Program

It was mentioned previously that **metrics** are part of the maintenance model -- they assist in evaluating the effects of change. When used over hundreds of components, the **metrics** can assume numerical values (e.g., for Completeness: ratio of completed components to total number of components in the system). For a single component, as in the example, a qualitative interpretation is appropriate. This is done below for the example, using typical metrics. Although the modification has improved **functionality**, it has degraded **maintainability**.

TABLE 2

METRICS APPLIED TO EXAMPLE PROGRAM

Metric	Original Program	Modified Program
Completeness:		
Are all required program parts present?	Yes	No.The correct comment is missing.
Consistency:		
Are the code and documentation uniform and free of contradiction?	Yes	No. The comment contradicts the commands and vice versa.
Modularity:		
Is the structure cohesive and self-contained?	No	No. Quirks of the DOS language inhibit modularity, but similar commands are grouped.
Traceability:		
Can the program parts be traced from one to another?	Yes	No. Can't trace between commands, drives and directories.
Verifiability:		
Can the correct operation and performance of the program be verified?	Yes	No. The erroneous comment confuses the verification.

B. Characteristics of Programming Language

Characteristics of the programming language can also significantly influence the ability to maintain [9]. Two brief examples from the DOS language [10] will be given:

o PATH command: If this command appears once and is repeated, the most recent occurrence of the command is the only one in effect. This means that any paths used to establish directories in a previous occurrence are lost unless they are repeated in the new PATH command. In effect, this means that a new path must be a superset of the previous path, if all original directory information is to be retained. However, this could result in long path commands and, without writing complicated logic, commands are limited to a single line! Thus the maintenance principle of being able to make a selective change (i.e., one wants to just add or delete parts of the PATH command, not write a new one) cannot be achieved with this command.

o IF command: The IF command has the format: IF string1==string2 command. The requirement for the second `=' is unexpected. This nuance of the language has caused several errors in implementing network batch files. This seemingly minor item can cause havoc in maintenance because a frequent change to batch files occurs as the result of adding capabilities to the network that are conditioned on the availability of certain resources. The IF command is key to specifying these conditions.

X. FURTHER RESEARCH

Further research is necessary to examine development methodologies in more detail with respect to their influence on maintainability, for example the object oriented approach [9]. The objectives of this paper have been to make a start towards the goal of standardizing maintenance by proposing that a change management methodology is the key to standardization, and to begin a dialogue with the software engineering community concerning approaches for standardizing maintenance. The objective has not been to solve the whole problem, which is complex.

XI. SUMMARY

We have contrasted software maintenance with hardware maintenanc Although there are similarities, the major difference -- the ease changing software -- causes unique software maintenance problems. We ha proposed that maintenance can be improved through standardization. T elements of the proposed standardization process are the following:

- o Metrics
- o Model of maintenance
- o Change documentation
- o Software communication mechanisms
- o Development methodology supportive of maintenance

An example was presented of the application of one developme methodology -- state diagrams -- to illustrate how proposed a accomplished changes can be illuminated so that errors can be avoided a maintainability improved.

Finally, we stated that because the maintenance problem is so comple more research must be done -- particularly on the relationship betwe development methodologies and maintainability -- before maintenance can standardized. However, we feel that the first four elements -- metric model of maintenance, change documentation, and software communicati mechanisms -- have merit and that NAVMASSO should evaluate them possible adoption.

XII. REFERENCES

[1] An American National Standard IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Standard 729, 1983.

[2] Bennet P. Lientz and E. Burton Swanson, "Problems in Application Software Maintenance", Comm. ACM, vol. 24, no. 11, pp. 763-769, Nov. 1981.

[3] Bennet P. Lientz and E. Burton Swanson, "Software Maintenance Management", Reading, MA: Addison-Wesley Publishing Co., 1980.

[4] Norman F. Schneidewind, "The State of Software Maintenance", Trans. on Soft. Engr., vol. Se-13, no. 3, pp. 303-310, March 1987.

[5] Meir M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution", Proc. of the IEEE, vol. 68, no. 9, pp. 1060-1076, September 1980.

[6] S. Letovsky and E. Soloway, "Strategies for Documenting Delocalized Plans", Proc. of the Conference on Software Maintenance - 1985, Computer Society Press, pp. 144-151.

[7] Rome Air Development Center, RADC-TR-85-37, Final Technical Report, February 1985.

[8] Bob Britcher and Jim Craig, "Upgrading Aging Software Systems Using Modern Software Engineering Practices: IBM-FSD's Conversion of FAA's National Airspace (NAS) En Route Stage A Software From 9020s to S/370 Processors', Proceedings, Conference on Software Maintenance-1985, Computer Society Press, pp. 162-170.

[9] Grady Booch, Software Engineering with Ada, The Benjamin/Cummings Publishing Company, Inc., 1983.

[10]Disk Operating System Reference Version 3.2, IBM Corp. and Microsoft, Inc., February 1986.

[11]IBM PC Local Area Network Program User's Guide Version 1.10, February 1986.

[12]Harlan D. Mills. "Stepwise Refinement and Verification in Box-Structured Systems", Computer, vol. 21, no. 6, June 1988, pp. 23-36.

DISTRIBUTION LIST

Mr. Roger Daugherty, Code 01B	1
Navy Management Systems Support Office	
Naval Air Station	
Building R52-7	
Norfolk, VA 23511-6694	
Commanding Officer	
Navy Management Systems Support Office	1
Naval Air Station	
Norfolk, VA 23511-6694	
Technical Director	
Navy Management Systems Support Office	1
Naval Air Station	
Norfolk, VA 23511-6694	
Prof. Tarek Abdel-Hamid	1
Code 54Ah	
Naval Postgraduate School	
Monterey, CA 93943	
Prof. Norman Schneidewind	10
Code 54Ss	
Naval Postgraduate School	
Monterey, CA 93943	
National Technical Information Center	2
Cameron Station	
Alexandria, VA 23314	
Knox Library	2
Code 0142	
Naval Postgraduate School	
Monterey, CA 93943	
Computer Center Library	1
Code 0141	
Naval Postgraduate School	
Monterey, CA 93943	
Administrative Sciences Department Library	1
Code 54	
Naval Postgraduate School	
Monterey, CA 93943	
Research Administration	1
Code 012	
Naval Postgraduate School	
Monterey, CA 93943	